Int. J. on Recent Trends in Engineering and Technology, Vol. 9, No. 1, July 2013

# Optimized Software Implementation of ECC over 192-bit NIST Curve

Ravi Kishore Kodali[1], Harpreet Singh Budwal and Prof. Narasimha Sarma, N.V.S.
[1] National Institute of Technology, Warangal /Department of E and CE, Warangal, India
Email: ravikkodali@gmail.com

***Abstract*— RSA is the most widely used public key cryptosystem, serving the purpose of key exchange for symmetric key cryptography and authentication. Improved network security demands forward secrecy, which RSA is unable to provide. Also the lower per bit security of the RSA technique, makes it difficult to be implemented in resource constrained devices. The protocols including Elliptic curve Diffie-Hellman Ephemeral (ECDHE) as the key exchange mechanism and RSA for authentication, overcomes the drawback incurred by the RSA alone and provides forward secrecy. The advantage of forward secrecy in a network is accompanied with higher complexity and computational cost. This paper describes the complete optimized software implementation of elliptic curve over the NIST prime field. The arithmetic operations over the prime are discussed. Different coordinate systems for elliptic curve point representation like affine, projective, Jacobian projective, and mixed coordinate systems are elaborated. Various techniques for scalar multiplication like Binary, NAF, sliding window, fixed based window, comb method, are given. Scalar multiplication is the most dominating operation in Elliptic curve cryptography (ECC) which consumes 85% of the execution time. A controller based on the fuzzy logic is presented for an optimum selection of window width, w, in the scalar multiplication methods. A comparison of various techniques and combinations of different techniques to perform complete ECDHE operation are provided along with the implementation timings.**

***Index Terms*—forward secrecy, coordinate systems, scalar multiplication, ECDHE, Fuzzy controller**.

## I. Introduction

The Transport layer security (TLS) takes care of the security issues over a network. It mostly uses RSA or Diffie-Hellman (DH), either of them as its key exchange mechanism. RSA is the most widely used key exchange mechanism, primarily used for the key exchange in SKC type communication as the DH based key exchange is more expensive. In RSA, one pair of keys namely, public key and private key are used during one session, which lasts approximately for one month. If this pair of keys is compromised in near future, all the information encrypted during that session of SKC can be compromised. This is called as forward secrecy and the RSA is incapable to provide the same [1].

In other words, forward secrecy means that the information, which is secure in present will also be secure in future. The forward secrecy strength depends on the Discrete Logarithmic Problem (DLP) of DH key pair [2].

The drawback of high computational cost and complexity with DH_RSA approach can be overcome by making use of elliptic curve (EC) based DH. Table I shows that the same standard of security with reduced number of bits is pursued by the Elliptic Curve Cryptography (ECC) compared to DH. In ECDH-Ephemeral (ECDHE) technique, as shown in Fig. 1, the generated key pair is used for a single session, usually lasting for a short duration.

Table I. Comparable Key Sizes [3]

| DH | 1024 | 2048 | 3072 | 7689 |
|---|---|---|---|---|
| ECC | 163 | 233 | 283 | 409 |

A standard elliptic curve E, specifically for the purpose of cryptography over the prime field $(F_P)$ is given as:

$$y^2 \ mod \ p = (x^3 + ax + b) \ mod \ p, \quad (1)$$

where a,b $\in F_P$ and $(4a^3 + 27b^2) \ mod \ p \neq 0$ [4]. The points on E, are calculated using equation (1). Addition of two points (Point Addition) and doubling of a point (Point Doubling) are considered to the basic operations on EC. The mathematical formulae for point addition and point doubling are shown in Table II.

Table II. EC Mathematical Operations [4]

| EC Operations | Slope(s) | $X_3$ | $Y_3$ |
|---|---|---|---|
| Point Addition $P(X_1, Y_1) + Q(X_2, Y_2)$ | $\frac{y_2 - y_1}{x_2 - x_1}$ | $S^2 - x_1 - x_2$ | $S(x_1 - x_3) - y_1$ |
| Point Doubling $2P(X_1, Y_1)$ | $\frac{3x_1^2 + a}{2y_1}$ | $S^2 - 2x_1$ | $S(x_1 - x_3) - y_1$ |

The rest of the paper is organized as follows: Section II discusses about the arithmetic operations over the prime field, Section III briefs about the different coordinate systems for point addition and point doubling operation, Section IV introduces various techniques for scalar multiplication, Section V provides a comparison of different techniques and timing results and Section VI gives the conclusion for the techniques discussed.

## II. Prime Filed Arithmetic

This section describes the different arithmetic operations used in the $F_P$ during the software implementation of EC over NIST prime field. For the same, processor architecture of 32-bit is considered for having uniformity during imple




DOI: 01.IJRTET.9.1.524

mentation.

### A. Field Representation

The elements, within the prime field $F_P$, are the integer between [0 to P-1]. For the prime field P, m = log2(P) shows the number of bits and t = (m/32) shows the size required by integer (4 byte) array to store the elements in $F_p$. Prime field elements are stored in an array of size t as:

$$a = (a_{t-1}, a_{t-2}.....,a_1, a_0)$$

### B. Addition and Subtraction

The arithmetic addition operation in prime field is represented as, $c = (a + b)\ mod\ P$, where a, b and c are elements in the prime field. If the addition of exceeds the P, it is brought back within the prime field. The modular subtraction operation is also performed on the same lines. Table III shows the steps for Modular addition and subtraction operations.

TABLE III. MODULAR ADDITION AND MODULAR SUBTRACTION

| | Modular Addition | Modular Subtraction |
|---|---|---|
| Input | a, b ∈ [0, P-1] | a, b ∈ [0, P-1] |
| Output | C= (a + b) mod P | C = (a - b) mod P |
| 1.$C_0$ | Add( $a_0$, $b_0$) | Sub($a_0$, $b_0$) |
| 2.For i: 1 to t | $C_i$ = ADC ($a_i$, , $b_i$) | $C_i$ = SBB ($a_i$, , $b_i$) |
| 3.If carry =1 | $c \leftarrow c - p$ | $c \leftarrow c + p$ |
| 4.Return | $C$ | $C$ |

### C. Integer Multiplication

The integer multiplication operation in prime field is much more cost computational than addition and subtraction operations. Algorithm I gives the steps for multiplication, which makes use of 32*32 bit mul instruction and produces a 64-bit result. $d_0$, $d_1$, $d_2$ are the variable of size 32-bit, a and b array of size t and C is an array of size (2t-1). The result generated by Algorithm I is of the size 2m-bits, which is brought inside the prime field (reduced to m-bits) with the help of modular reduction Algorithm [5].

### D. Inversion

Among all the arithmetic operations over the prime field, inversion operation is the most expensive one. It is approximately 80 times more expensive than integer multiplication operation. Extended Euclidean Algorithm (EEA), as shown in Algorithm II, is used for calculating the inversion of the number in a prime field.

**Algorithm I:** Integer Multiplication [5]

Input: a, b ∈ [0, P-1]
Output: C = a * b
1: Initialize variable $d_0, d_1, d_2$ to zero
2: **For** k: 0 to 2(t-1) **do**
3: **For** i: 0 to t **do**
4: **For** j: 0 to t **do**
5: If ( (i + j) = = k)
6: R = $a_i$ * $b_i$ where R is array of size 2
7: $d_0 \leftarrow$ ADD ($d_0$, R[0])
8: $d_1 \leftarrow$ ADC($d_1$ , R[1])
9: $d_2 \leftarrow$ ADC($d_2$ , 0)
10: **end For**
11: **end For**
12: $C_k \leftarrow d_0, d_1 \leftarrow d_2, d_2 \leftarrow 0$
13: **end For**
14: $C_{2k-1} \leftarrow d_0$
15: Return C

**Algorithm II** Binary Inversion Algorithm [5]

Input: a ∈ [0, P − 1], prime P
Output: $a^{-1}$ mod P
1: i ← a, j ← p, D ← 1, C ← 0
2: **while** i ≠ **do**
3: **while** i is even **do**
4: i ← i/2 , If D is even then D ← D/2 else D ← (D + P)/2
5: **end while**
6: **while** j is even **do**
7: j ← j/2, If C is even then C ← C/2 else C ← (C + p)/2
8: **end while**
9: If i ≥ j then i ← i − j, D ← D − C
10: Else j ← j − i, C ← C − D
11: **end while**
12: Return C

## III. ELLIPTIC CURVE CRYPTOGRAPHY

The classical Elliptic Curve Diffie Hellman ephemeral (ECDHE) scheme is illustrated in Fig. 1.

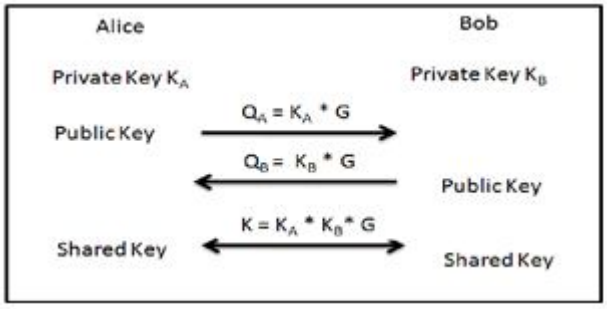


Figure.1. Elliptic Curve Diffie-Hellman

Initially, both the server and the client nodes agree on a particular Elliptic curve (EC) over some prime field, $F_P$, with a specific base point, G, termed as the generator point. G is one of the valid points on the EC curve, which has highest order [4]. Both the server and client nodes set their respective private keys by selecting randomly any scalar integer from the prime field . The corresponding public keys, $Q_A$ and $Q_B$, are computed by multiplying the generator point, G, with the corresponding private keys namely and . These public keys are then shared over the network between the server and the client, which again multiply them with their corresponding private key, hence generating a shared secret key given as: $T = K_A * Q_B = K_B * Q_A$. Due to Elliptic Curve Discrete Logarithmic Problem (ECDLP), even though the value of $Q_A$, $Q_B$, and G are spread over the network, it


DOI: 01.IJRTET.9.1.524

would be computationally infeasible to calculate the private keys $K_A$ and $K_B$ for an intruder [6].

The two frequently used operations in ECDHE key exchange are: scalar multiplication and modular reduction. Scalar multiplication based on ECDLP, consumes 85% of computational cost in ECC [7]. It consists of multiplying a point on the EC, with a scalar integer k, such that k $\in F_P$. Scalar Multiplication comprises of successive summation of point addition and point doubling operations.

Thus the speed of ECDHE key exchange method is directly proportional to the performance of the scalar multiplication on the EC. The efficiency of the scalar multiplication can be improved by using a recoding algorithm for the integer, k described in section IV and using different coordinate systems for point addition and doubling operations described in section III A and B.

### A. Coordinate System

The point addition (PA) and point doubling (PD) operations, as shown in Table I, make use of the affine coordinate system. In affine representation, points on the EC consist of x and y coordinates. The number of prime field arithmetic operations used by PA and PD operation in affine coordinate system is given in Table IV. As discussed in section II, high execution time of inversion operation in $F_P$ decreases the efficiency of PA and PD operation.

TABLE IV. EQUIVALENT PRIME FIELD OPERATION

| | Inversion | Multiplication |
|---|---|---|
| Point Addition (P $\neq$ Q) | 1 | 3 |
| Point Doubling (P = Q) | 1 | 4 |

### B. Projective Coordinate System

The advantage of representing EC point in this coordinate system is that it does not involve inversion operation, while performing PA and PD operations. This gives an edge to use this system over affine coordinate system. Many variations of projective coordinate system have been introduced, and out of them the most efficient ones are presented here. The first one is Standard projective system (P), in which point is represented as (x : y : z) $z \neq 0$, whose Affine coordinate (A) equivalent is given by $(x/z,\ y/z)$ and the equation of curve is given as: $y^2z = x^3 - 3xz^2 + bz^2$. In Jacobian projective system (J), a point is represented as (x : y : z) $z \neq 0$, whose Affine equivalent is given by $(x/z^2, y/z^3)$ and the equation of curve is given as: $y^2 = x^3 - 3xz^4 + bz^6$. In Chudnovsky Jacobian coordinate system (C), the Jacobian point, (x : y : z) is represented as (x : y : z : $z^2$ : $z^3$) [5].

The formulae for PA and PD, which do not involve inversion operation in different variations of projective system can be obtained by substituting the respective point into it affine equivalent in Table I. The number of prime field operations required by different coordinate systems to carry out PA and PD is given in Table V.

It can be observed from Table V that PD in Jacobian coordinate system and PA in mixed coordinate system (Jacobian + affine) leads to minimum number of prime field operations. This will improve the overall efficiency of ECC.

TABLE V. OPERATION COUNT FOR ELLIPTIC CURVE PA AND PD

| Doubling | | Addition | | Mixed | |
|---|---|---|---|---|---|
| 2A→A | 1I, 2M, 2S | A+A→A | 1I, 2M, 1S | J+A→J | 8M, 3S |
| 2P→P | 7M, 3S | P+P→P | 12M, 2S | J+C→ J | 11M, 3S |
| 2J→J | 4M, 4S | J+J→J | 12M, 3S | C+A→C | 8M, 3S |
| 2C→C | 5M, 4S | C+C→C | 11M, 3S | | |

The mathematical steps for calculating PD in Jacobian coordinate system $2(x_1 : y_1 : z_1) = (x_3 : y_3 : z_3)$ are:

$$A = 4x_1.y_1^2,\ B = 8y_1^4,\ C = 3(x_1 - z_1^2).(x_1 + z_1^2)$$

$$D = -2A + C^2,\ x_3 = D,\ y_3 = C.(A - D) - B$$

$$z_3 = 2y_1.z_1$$

The mathematical steps for calculating PA in Mixed coordinate $(x_1 : y_1 : z_1) + (x_2 : y_2 : 1) = (x_3 : y_3 : z_3)$ system are:

$$A = x_2.z_1^2,\ B = y_2.z_1^3,\ C = A - x_1,$$

$$D = B - y_1,\ x_3 = D^2 - (C^3 + 2.x_1.C^2),$$

$$y_3 = D.(x_1.C^2 - x_3) - y_1.C^3,\ z_3 = z_1.C$$

## IV. SCALAR MULTIPLICATION

Scalar multiplication on EC is represented as: $T = K * G$, where T, G are the points on the EC and k is an integer in the prime field. Different scalar multiplication algorithms, recoding the integer, k, to reduce the number of one's in it in order to reduce its hamming weight (W). This reduces the number of PA operations required for scalar multiplication, hence speeding up the operation. The scalar multiplication algorithms can be classified into two categories depending on G: Unknown point and Fixed point.

Scalar multiplication with unknown point: algorithm belonging to this category is used when G is not known a priori. This includes Binary, NAF, Sliding window and NAF sliding window method.

### A. Binary Method

Binary method is the simplest and the most computationally expensive scalar multiplication method [8]. Binary representation of the integer k helps users to conclude that, consecutive summation of the point doubling and point addition operations over the EC leads to scalar multiplication.

$$k = \sum_{j=0}^{l-1} K_j 2^j. \quad (2)$$

$$Q = kG = K_0G + 2(K_1G + ... + 2(K_{l-1}G)). \quad (3)$$

In scalar multiplication, point addition (A) and point doubling (D) operations are used to determine computational





DOI: 01.IJRTET.9.1.524

cost of different algorithms. The number of 1's in the binary representation of k, is called its Hamming weight (W) and l, is the total number of bits in k. The computational cost of the Binary method is given by equation (4).

$$Cost = (W-1)A + (l-1)D. \quad (4)$$

*B. NAF Method*

Contrary to the representation of k in Binary method, if the representation of k also consists of negative bits, i.e. {-1, 0, 1}, then it is called as Binary Signed Digit Representation (SDR). Non-adjacent form (NAF) is a specific case of SDR. In NAF, both W and l are kept as small as possible. An NAF of a positive integer, k, is given by the equation (5) [8].

$$k = \sum_{j=0}^{l-1} K_j 2^j, \quad K_j \in \{-1, 0, 1\}. \quad (5)$$

In NAF of k, multiplication of any two consecutive bits is always zero i.e. $K_j * K_{j+1} = 0$. The NAF form of integer, k, is denoted as NAF (k), its length is at most (l+1) of the binary form of k. Algorithm III is used to convert the integer k into its NAF (k).

Scalar multiplication for NAF (k) is obtained using equation (3), the only difference is that when (-1) appears G should be subtracted from Q. The W of the integer, k, is approximately reduced to (l/3) by using NAF (k) and the number of point doubling operations remains to be the same as in the Binary method [8]. The computational cost of the scalar multiplication using NAF (k) is given by the equation (6).

$$Cost = \frac{l}{3}A + lD \quad (6)$$

**Algorithm III** Binary (k) to NAF (k) [9]

Input: k ∈ [0, P-1]
Output: K = NAF (k)

1: i ← 0
2: **while** k ≥ 1 **do**
3: If k is odd then
4: $K_i \leftarrow (2 - (k \bmod 4))$, k ← k − $K_i$
5: Else $K_i \leftarrow 0$
6: k ← k/2, i ← i + 1
7: **end while**
8: Return K

Table VI illustrates different examples of NAF (k).

TABLE VI. NAF FORM OF INTEGERS

| Decimal Representation | Binary Representation | NAF Representation |
|---|---|---|
| 26 | 11010 | $10\bar{1}010$ |
| 1122334455 | 1000 0101 1100 1010 1110 1101 1110 111 | $10001010\bar{1}0\ 0\bar{1}01\ 0\bar{1}0\bar{1}\ 000\bar{1}00\bar{1}0000\bar{1}001$ |

*C. Sliding Window Method*

To reduce the computational cost of Binary and NAF methods, the digits used for representing k, can be extended beyond 3 bits as in NAF i.e. {-1, 0, 1}. This reduces the number of point additions. But the advantage comes at a cost, a small number of values, that are multiple of G, should be pre-computed and stored in memory, such that they are added or subtracted to the Q [8] during multiplication. The memory required to hold these pre-computed values becomes a constraint. The sliding window method processes at most consecutive w digits of the scalar integer, k, such that the decimal equivalent of the window-w consecutive digits should always be odd. The method has no fixed window width- w, it can be varied from 1 to w depending on the number of zero's at the LSB bit. Algorithm IV presents the scalar multiplication for the sliding window method with binary representation of integer k. Table VI provides the details for the different window widths -w. The computational cost for the binary sliding window method is shown in Table VII, where V (w) as given in the equation (7), is the average length of a run of 0's within the window [4].

$$V(w) = \frac{4}{3} - \frac{(-1)^w}{3 * 2^{w-2}} \quad (7)$$

*D. NAF Sliding Window Method*

Algorithm V uses both sliding window method and NAF (k). The NAF (k) is computed and the same is given as input to this algorithm.

The combination of sliding window and NAF method, reduces the number of pre-computations required compared to the combination of binary sliding window methods. This improves the efficiency of the algorithm, in a system with

**Algorithm IV** Binary Sliding window for scalar multiplication [10]

Input: Generator point G, k, window w
Output: Q = k*G

1: Calculate [x] G where x = 1, 3, 5...., $(2^{(w-1)} - 1)$
2: j ← l − 1, where l is length of k
3: **while** j ≥ 0 **do**
4: if ($K_i$ = = 0)
5: Q ← [2]Q , N ← 0, j ← j − 1
6: end if
7: Else
8: i ← maximum (j − w + 1, 0)
9: **while** $K_i$ = = 0 **do**
10: i ← i + 1
11: **end while**
12: **For** d = 1 to (j − i + 1) **do**
13: d = d + 1 and Q ← [2]Q
14: **end For**
15: N ← $(K_j.....K_i)_2$
16: j ← i − 1
17: end else
18: Q ← Q ⊕[N ]G
19: **end while**
20: Return Q




DOI: 01.IJRTET.9.1.524

TABLE VII. DIFFERENT WINDOW WIDTHS COMPARISON FOR SLIDING WIDOW METHOD

| Window Width-w | Number of Pre-computations | Integer K=2973 | Intermediate Value | Number of addition | Number of doubling | Pre-computation |
|---|---|---|---|---|---|---|
| 3 | 3 | 101 11 00 111 01 | 5G, 10G, 20G, 23G, 46G, 92G, 184G, 368G, 736G, 743G, 1486G, 2972G, 2973G. | 3 | 9 | [3]G, [5]G, [7]G |
| 5 | 15 | 10111 00 111 0 1 | 23G, 46G, 92G, 184G, 368G, 736G, 743G, 1486G, 2972G, 2973G. | 2 | 7 | [3]G, [5]G, [7]G [9]G..... [25]G [27]G, [29]G, [31]G |

**Algorithm V** NAF Sliding window for Scalar Multiplication [4]

Input: Generator Point G, integer k, window w
Output: Q = k*G

1: Compute NAF(k) with Algorithm 3.
2: Calculate [x]G, where x={1, 3, 5, ,.....,$(2^w - (-1)^w/3) - 1$}
3: j ←l − 1 where l is the length of k
4: **while** j ≥ 0 **do**
5: Algorithm (VI) Steps 4 to 17
6: If (N ≥ 0)
7: Q ← Q + [N ]G
8: Else Q ← Q − [N ]G
9: **end while**
10: **return** (Q)

limited memory.

The computational cost for the NAF and sliding window method is given in Table VII. The computational cost of the Algorithms 4 and 5 depends on the window width, w. An optimal window width, w, needs to be chosen beforehand, in order to reduce the computational cost.

TABLE VIII. COMPUTATIONAL COST FOR SLIDING WINDOW SCALAR MULTIPLICATION

| Method [4] | Number of Doubling(D) | Number of Addition (A) | Number of Pre-computation |
|---|---|---|---|
| Binary | $l$ | $\frac{l}{w+v(w)}$ | 1D+$(2^{w-1}-1)$ |
| NAF | $l$ | $\frac{l}{w+v(w)}$ | 1D + $(2\frac{2^w-(-1)^w}{3}-1)$ |

Scalar multiplication with fixed point**:** algorithms in this category are used, when the point, G is known a priori. This includes Fixed based window and Fixed based comb methods.

*E. Fixed Based Window Method*

As the generator point, G, is already known, more number of pre-computed values of G can be stored in the memory. Therefore, higher window width-w is considered as compared to NAF sliding window method. In this technique, w consecutive bits of k are combined and k can be represented as:$k = (K_{d-1}, ....., K_1, K_0)_{2^w}$ where d = (L/w). The pre-computations required are: $2^{wi}$ where 0 > i > (d - 1). The Algorithm VI shows scalar multiplication using Fixed Based window Method.

As observed from Algorithm VI, it consists of only point addition operation, while performing scalar multiplication but the number of PD and PA operation required during the pre-computation are high. The computational cost of Fixed based window method is given by using equation (8)

$$Cost = (2^w - d - 3)A \quad (8)$$

**Algorithm VI** Scalar Multiplication using Fixed based comb method [4]

Input: Integer k and window w
Output: Q = k * P

1: pre-computed value of G
2: initialize variable t ← 0
3: **For** i : $2^w - 1$ to 1 **do**
4: for each j, if $K_i$ == i then t ← t + $2^{wi}G$
5: Q ← Q + t
6: **end for**
Return Q

*F. Fixed Base Comb Method*

In fixed based comb method, binary representation of k is modified by padding (d*w-t) zeros on the left side. The d consecutive bits in k are combined together, which is represented as: $k = (K^{w-1}, ...., K^0, K^1)$. To accelerate the computations, some values of based point are pre-computed, given as:

$[a_{w-1}, ...., a_2, a_1, a_0]G = a_{w-1}2^{(w-1)d}G + .... + a_2 2^{2d}G + a_1 2^d G + a_0 G$

Algorithm VII shows the comb method for scalar multiplication.

**Algorithm VII** Fixed Based Comb method for Scalar Multiplication [4]

Input: Integer k and window w
Output: Q = k*G;

1: pre-computed value of G
2: Modifying Integer k as:$k = (K^{w-1}, ...., K^0, K^1)$
3: Let $K_i^j$represents $ith$ bit of $K^j$




DOI: 01.IJRTET.9.1.524

4: Q ← 0̂
5: **For** i: d-1 to 0 **do**
6: Q ← 2Q
7: Q ← Q + $[K_i^{w-1}, ..., K_i^1, K_i^0]$G
8: **end For**
9: Return Q

This method proves to be very efficient, as it uses only (d-1) doublings and also the number of pre-computed values is comparatively less. The computational cost for this method is calculated with the help of equation (9).

$$cost = \left(\frac{2^w-1}{2^w}d - 1\right)A + (d-1)D \quad (9)$$

## V. Results And Comparisons

In this section, the results obtained after the software implementation of the various techniques introduced in this paper are discussed. A Linux platform, with gcc compiler, is used for the implementation. The 192-bit prime field, P and the domain parameters, a and b used for the software implementation are given in Table VIII.

Table IX. Domain parameter for 192-bit NIST prime field

| Parameter | 192-bit value (hex) |
|---|---|
| $P_{192}$ | Ffffffffffffffffffffffffffffffeffffffffffffffff |
| a | Ffffffffffffffffffffffffffffffeffffffffffffffffc |
| b | 64210519e59c80e70fa7e9ab72243049feb8deecc 146b9b1 |

The timing results for the arithmetic operations over the prime field, $F_P$, are given in Table IX. It can be observed that the inversion operation is much more time consuming than other operations.

Table X. Timings for prime field operations

| Operation | Timing (μs) |
|---|---|
| Addition | 5.44 |
| Subtraction | 2.015 |
| Multiplication | 17.28 |
| Inverse | 575.2 |

The timing results for point addition and point doubling operations over different coordinate systems is given in Table X. From Table X, it is proved that the most efficient implementation of point doubling operation is obtained in Jacobian coordinate system and the most efficient implementation of point addition operation is obtained in Jacobian + affine, namely mixed coordinate system.

Table XI. Timings for Point Addition and Point Doubling operation

| Coordinate System | Point Addition | Point Doubling |
|---|---|---|
| Affine | 0.598 ms | 0.597 ms |
| Jacobian | - | 0.129 ms |
| Jacobian + Affine | 0.132 ms | - |

The timing results for different algorithms of scalar multiplication methods and their comparison in terms of PA and PD operation are given in Table XI. In the comparison, mixed Jacobian (MJ) coordinate system, where PA uses mixed coordinate system and PD uses Jacobian coordinate system.

Table XII. Comparison of Different Scalar Multiplication Methods

| Method | PA | PD | Timing (second) |
|---|---|---|---|
| Binary(Affine) | 91 | 191 | 0.063 |
| Binary (MJ) | 91 | 191 | 0.0196 |
| NAF(Affine) | 64 | 191 | 0.057 |
| NAF (MJ) | 64 | 191 | 0.0179 |
| Sliding window (affine) | 36 | 189 | 0.050 |
| Sliding window (MJ) | 36 | 189 | 0.0156 |
| NAF sliding window ( Affine) | 30 | 189 | 0.048 |
| NAF sliding window (MJ) | 30 | 189 | 0.0152 |
| Fixed Based Comb (Affine) | 56 | 63 | 0.027 |
| Fixed Based Comb (MJ) | 56 | 63 | 0.008 |

## VI. Conclusion

In this work, all prime field arithmetic operations, different coordinate systems for EC point operation and different scalar multiplication methods are discussed and the results are compared. It is observed that, the inversion operation in a prime field has very high execution time compared to its counterpart. It infers that, such type of point addition and point doubling operation should be used, which does not involve inversion operation, so that over all execution time is less. Hence projective, Jacobian and chudnovsky coordinate systems are discussed. The Jacobian system for point doubling and Jacobian plus affine system for point addition proves to most efficient one. Different scalar multiplication methods for unknown point namely, Binary, NAF, Sliding window and NAF sliding window methods are discussed. The NAF sliding window method for a particular window width-w proves to outperform the remaining methods as it comprises of minimum number of point addition and point doubling operations. The fixed based window method and fixed based comb method are also discussed for fixed point scalar multiplication operation. The comb method proves to be much more efficient than the other. The NAF sliding window method for unknown point and Comb method for fixed point, with the point addition operation in mixed coordinated system and point doubling operation in Jacobian coordinate system leads to an optimized and efficient implementation of ECDHE.

## References

[1] B. Vincent, “Ssl/tls and perfect forward secrecy,” 2011 .
[2] E. Kasper, “Fast elliptic curve cryptography in openssl,” *Financial Cryptography and Data Security*, pp. 27–39, 2012.
[3] S. Blake-Wilson, B. Moeller, V. Gupta, C. Hawk, and N. Bolyard, “Elliptic curve cryptography (ecc) cipher suites for trans port layer security (tls),” 2006.
[4] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*. Springer, 2004.
[5] M. Brown, D. Hankerson, J. Lopez, and A. Menezes, *Software implementation of the NIST elliptic curves over prime field*,. Springer, 2001.
[6] R. K. Kodali and H. S. Budwal, “High performance scalar multiplication for ecc,” in *Computer Communication and Informatics (ICCCI), 2013 International Conference on*. IEEE,




DOI: 01.IJRTET.9.1.524

2013, pp. 1–4.

[7] N. Gura, A. Patel, A. Wander, H. Eberle, and S. Shantz, "Comparing elliptic curve cryptography and rsa on 8-bit cpu's," *Cryptographic Hardware and Embedded Systems-CHES 2004*, pp. 925–943, 2004.

[8] O. YAYLA, "Scalar multiplication on elliptic curves," Ph. D. dissertation, Master's Thesis, Department of Cryptography, Middle East Technical University, August 2006.http://www w3.iam.metu.edu.tr/iam/images/3/3e/O%C4%9Fuzyaylathesis. pdf, 2006.

[9] X. Huang and D. Sharma, "Fuzzy controlling window for ell iptic curve cryptography in wireless networks," in *Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on* IEEE, 2010, pp. 521–526.

[10] X. Huang, D. Sharma, and H. Cui, "Fuzzy controlling window for elliptic curve cryptography in wireless sensor networks," in *Information Networking (ICOIN), 2012 International Conference on*. IEEE, 2012, pp. 312–317.


DOI: 01.IJRTET.9.1.524